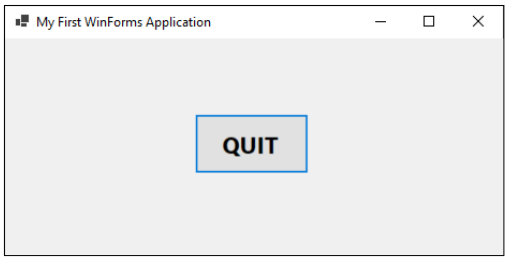
Eseményvezérelt alkalmazások: 3. gyakorlat

A munkafüzet bevezet a Windows Forms alapú asztali grafikus alkalmazás fejleszésbe, Visual Studio 2022 fejlesztőkörnyezetet használva. Megismerjük a VS grafikus tervezőfelületét, több alapvető felületi vezérlőt. Alkalmazásainkat az eseményvezérelt paradigma mentén készítjük el.

1 Első grafikus asztali alkalmazás ^{KM}

Készítsünk egy olyan alkalmazást, amely egy nyomógombot jelenít meg egy ablakban. Erre rákattintva az alkalmazás álljon le.



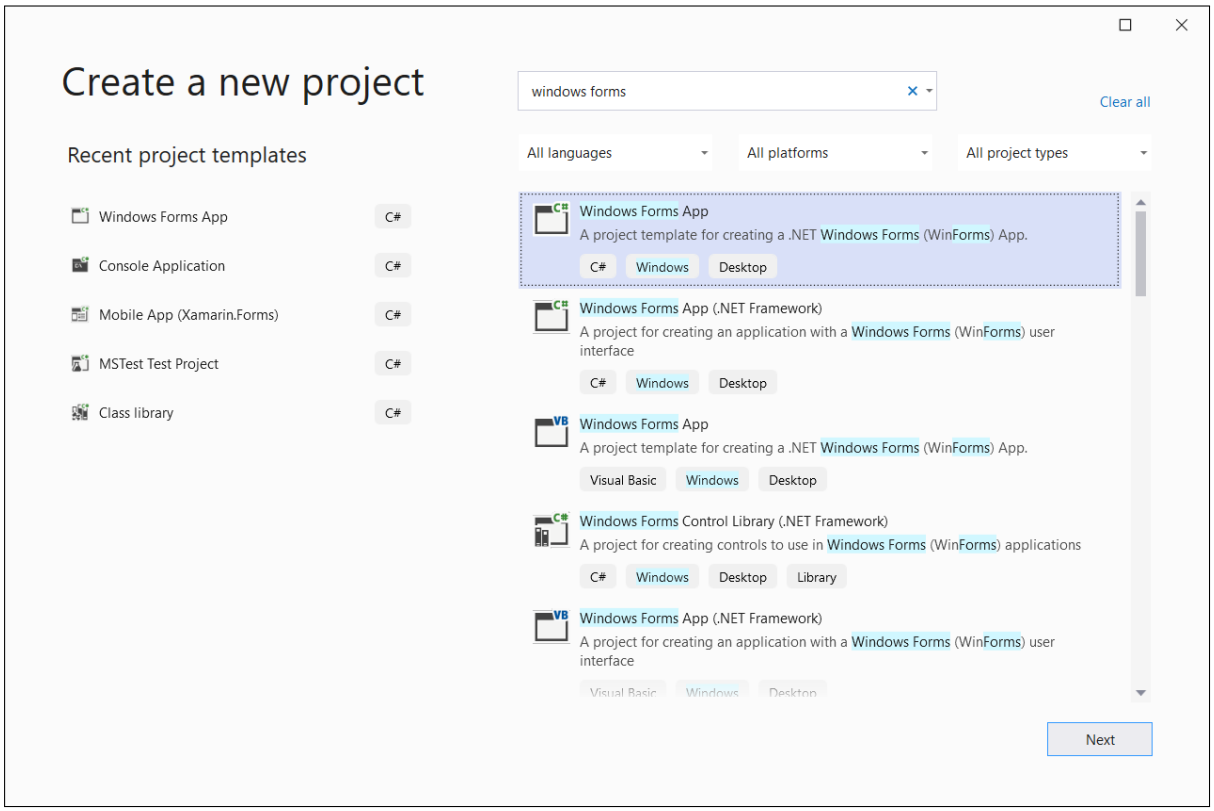


Figure 1: Windows Forms projekt létrehozása

A Visual Studio 2022 elindítása után válasszuk a **Create a new project** menüpontot.

A következő ablakban válasszuk ki a **Windows Forms App** projektsablont, a projektnek adjunk tetszőleges nevet (pl. *Quit*).

> Ne a *Windows Forms App (.NET Framework)* sablont válasszuk, ugyanis az .NET 8 helyett a korábbi .NET Frameworkre épülő projektet fog létrehozni.

A generált kód egy `Form`-ból (`Form1`) és egy `Program` nevű osztályból áll, ez utóbbiban található a `Main` metódus. A `Form1` az alkalmazásunk elsődleges ablaka. A `Form1`-re kétszer kattintva előhozhatjuk a tervező (*Designer*) nézetet, ahol manuálisan helyezhetünk fel vezérlőket az ablakra. A `Form1.cs`-t lenyitva láthatjuk a Designer mögött lévő generált kódot (`Form1.Designer.cs`), ebbe ne nyúljunk bele. A `Form1` szerkesztendő kódját a `Form1.cs`-re jobb klikkelve, a *View code* menüponttal hozhatjuk elő. A továbbiakban ebben a fájlban és a Designer nézetben fogunk dolgozni.

A Windows Forms Designer fontosabb paneljei a következők:

- Középen kódszerkesztő helyén az alkalmazás módosítható grafikus tervét láthatjuk, ami kezdetben egy üres alkalmazás ablak. A vezérlőket itt pozícionálhatjuk és méretezhetjük is.
- Bal szélen a *Toolbox* panel a felhelyezhető Windows Forms vezérlőket találjuk, amelyek egyszerű *drag-and-drop* módszerrel hozzáadhatóak az alkalmazáshoz.
- Jobb szélen a *Solution Explorer* alatt a *Properties* panel tartalmazza a kijelölt vezérlő (vagy ablak) tulajdonságait és eseményeit, amelyeket itt változtatni is lehet.

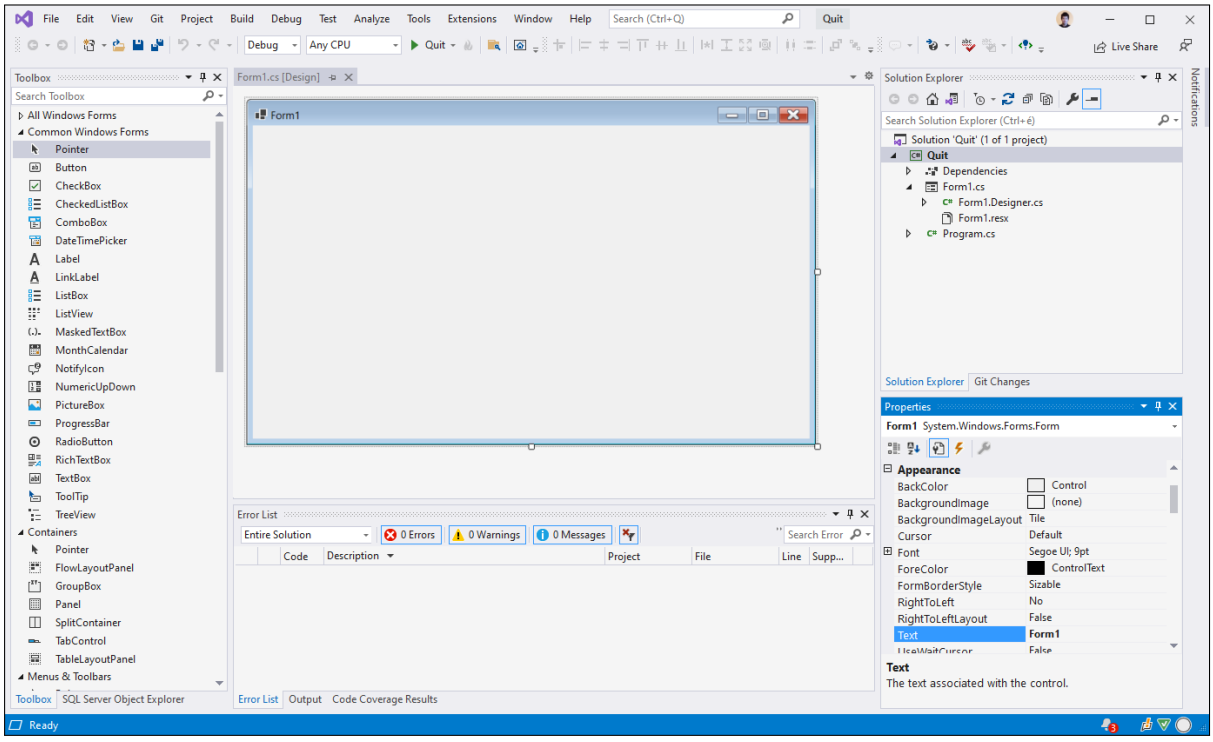


Figure 2: Windows Forms Designer

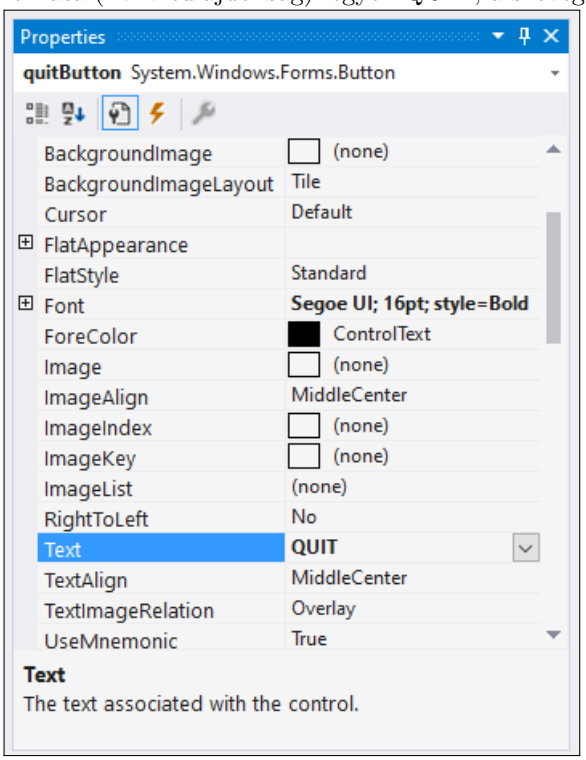
Az egyes panelek áthelyezhetőek a Visual Studion belül, illetve amennyiben valamelyiket bezárnánk, akkor a *View* menüpontból újra megnyitható.

1.1 Grafikus felület elkészítése

Végezzük el a következő feladatokat:

1. Módosítsuk a `Form1` ablak `Text` tulajdonságát *My First WinForms Application*-re.

2. Az ablak mérete legyen kisebb, azt a tervező felületén a jobb alsó sarkát megragadva csökkentsük; vagy a `Size` tulajdonság értékét módosítsuk a *Properties* panelen.
3. A *Toolbox*-ból adjunk hozzá egy `Button` vezérlőt (nyomógomb) ablakhoz. Méretét növeljük meg, a felirata (`Text` tulajdonság) legyen *QUIT*, a szöveg betűmérete legyen 16pt és félkövér.



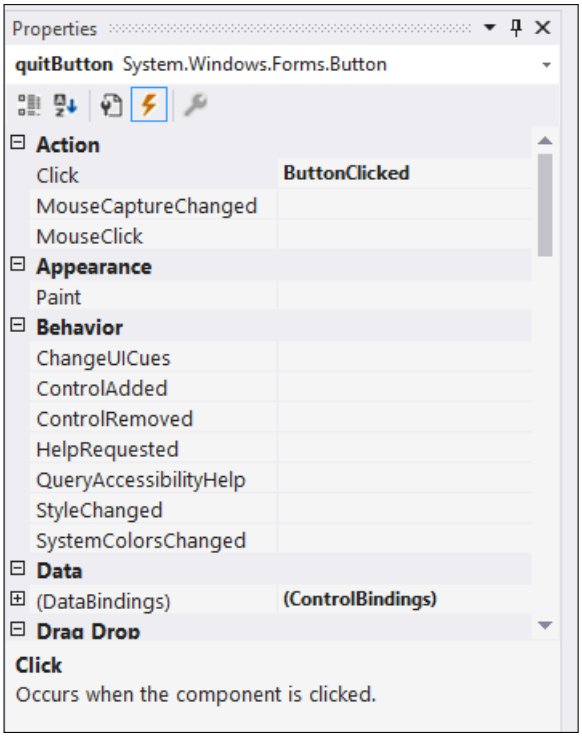
4. A nyomógomb `(Name)` tulajdonságát módosítsuk `quitButton`-ra. Ezzel valójában azt a változónevet is beállítottuk, amivel hivatkozhatunk az objektumra a forráskódból.

Az alkalmazást így már futtathatjuk a megszokott módon, a *QUIT* gomb kattintható is lesz, de a kattintás eseményére még semmi nem fog történni.

1.2 Eseménykezelés

A nyomógombon kiváltott eseményekhez eseménykezelőket lehet hozzárendelni, amelyet a Visual Studioban több féle módon is megtehetünk:

1. A Designerben duplán a nyomógombra kattintva a Visual Studio generál egy `quitButton_Click` eseménykezelő metódust a `Form1.cs` fájlba, amelynek törzsét megírhatjuk. A metódust fel is iratkoztatja a nyomógomb `Click` eseményére. (Ez a módszer csak a nyomógomb `Click` eseményére működik.)
2. A *Properties* panelen a villám ikonra kattintva a tulajdonságok helyett a kiválasztott vezérlő eseményeit tekinthetjük meg. A `Click` esemény mellé az eseménykezelő metódus preferált nevét beírva (pl. `ButtonClicked`), majd entert ütve, a Visual Studio elkészíti az eseménykezelő eljárást és feliratkoztatja az eseményre.



3. Amennyiben a `Form1` osztályunkban készítünk egy olyan eljárást, amelynek szignatúrája illeszkedik az esemény delegáltjához (`EventHandler`), akkor a *Properties* panelen az esemény melletti legördülő menüben fel fogja kínálni a Visual Studio.

```
public delegate void EventHandler(object sender, EventArgs e);
```

4. Valamint akár a forráskódban is elvégezhetjük az esemény és az eseménykezelő feliratkoztatását:

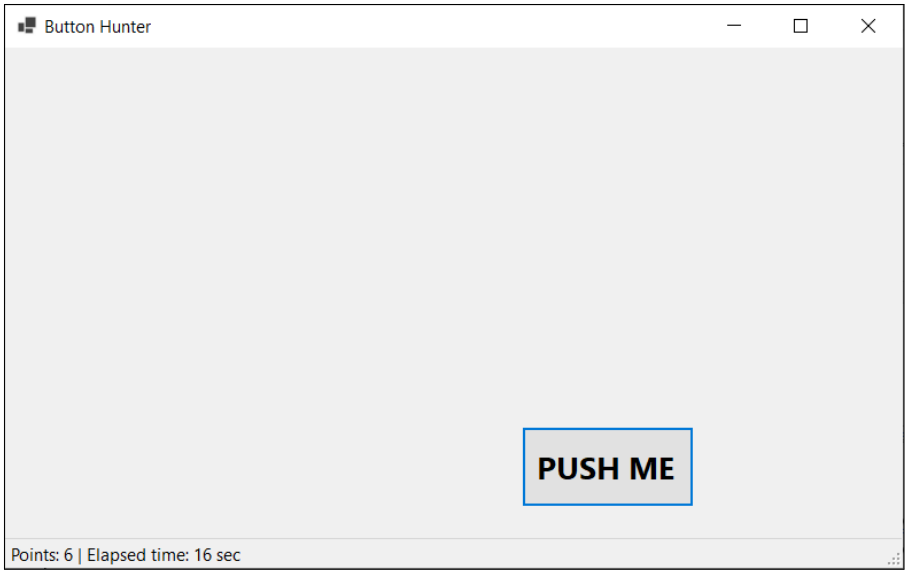
```
quitButton.Click += new EventHandler(ButtonClicked);
```

Fontos, hogy ez a generált kódot tartalmazó `InitalizeComponent()` metódus végrehajtása után tegyük meg, hiszen maga a nyomógomb csak ott kerül példányosításra, addig a `quitButton` értéke `null`.

Az eseménykezelő eljárásban zárjuk be az ablakot a `Close()` metódus meghívásával, amit a `System.Windows.Forms.Form` osztálytól megörökölt a `Form1` osztályunk. Mivel most ez az alkalmazásunk egyetlen ablaka, ezzel az alkalmazás futása is véget fog érni.

2 Gombvadászat *EM*

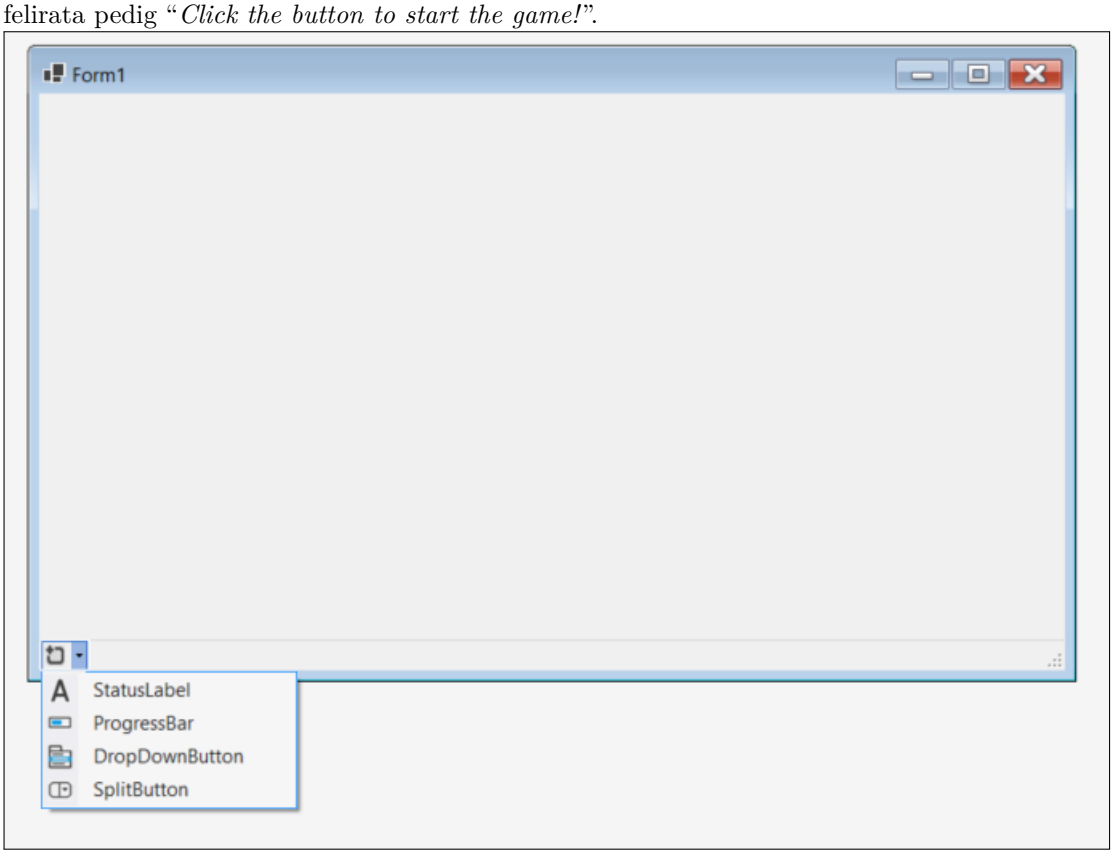
Készítsünk el egy egyszerű játékot. Az alkalmazás felületén egy nyomógomb jelenjen meg, amelyikre ha rákattintunk, az egy véletlenszerű másik pozícióba ugrik. A cél a gomb minél többször történő megnyomása. Írjuk ki a felület alsó sorában folyamatosan azt, hogy eddig hány kattintásnál járunk.



2.1 Grafikus felület elkészítése

Készítsünk egy új *Windows Forms App* projektet, amelynek a fő ablakát (`Form1`, de igény szerint át is nevezhető) a következő módon készítsük el:

- Az ablak felirata a címsorban (`Text` tulajdonság) legyen "*Button Hunter*", és a kiindulási méret (`Size` tulajdonság) mellett állítsunk be a felületnek egy értelmes minimális méretet (`MinimumSize`), hogy a nyomógombnak legyen helye majd ugrálni.
- Adjunk az ablakhoz egy nyomógombot (`Button`), amelynek a neve legyen `pushButton`, a felirata (`Text` tulajdonság) pedig "*PUSH ME*". A gomb és a felirat méretét növeljük meg.
- Vegyünk fel egy státusz sor vezérlőt is (`StatusStrip`), amelynek a neve legyen `statusStrip`. A státuszsorhoz adjunk egy címkét (`StatusLabel`), amelynek a neve `statusLabel` legyen, kezdeti felirata pedig "*Click the button to start the game!*".



2.2 Eseménykezelés

Adjunk hozzá a `Form1` osztályhoz két privát adattagot:

- egy `int points` mező tárolja a kattintások számát (pontok);
- egy `System.Random generator` mező a nyomógomb új pozíciójának (pszeudo-)véletlen előállítását fogja segíteni.

Az adattagokat inicializálhatjuk a konstruktorban vagy a deklarációval, pl.:

```
private int points = 0;
private Random generator = new Random();
```

Készítsünk eseménykezelőt a nyomógomb kattintásához (`Click` esemény). A nyomógomb új pozícióját a `Location` tulajdonsággal állítjuk be, és a koordinátákhoz a véletlenszámokat a `generator` objektumunkkal generáljuk. Ügyeljünk arra, hogy a gomb ne lóghasson le a felület központi területéről. Ehhez szükségünk van a gomb, a státuszsor és a központi terület magasságára és szélességére. Minden Windows Forms vezérlő rendelkezik a `Size` tulajdonsággal a méretének lekérdezéséhez (és beállításához), illetve a szélesség és a magasság közvetlenül a `Width` és `Height` tulajdonságokkal is elérhető. Az ablak esetén azonban a `Size` helyett a `ClientSize` tulajdonság ad most pontosabb eredményt: ez nem tartalmazza az ablak szegélyét és címsorát.

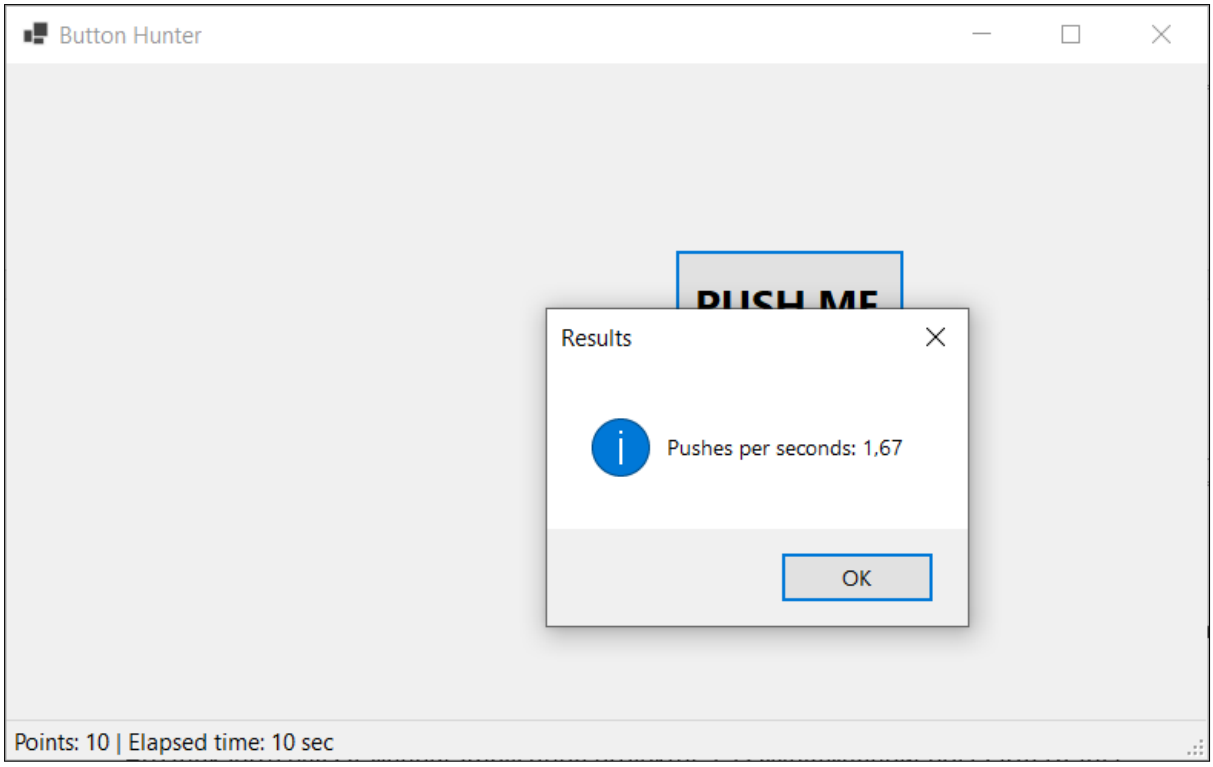
```
private void ButtonClicked(object sender, EventArgs e)
{
    int x = ClientSize.Width - pushButton.Width;
    int y = ClientSize.Height - pushButton.Height - statusStrip.Height;
    pushButton.Location = new Point(generator.Next(x), generator.Next(y));
    ++points;
    statusLabel.Text = $"Points: {points}";
}
```

Végezetül növeljük meg eggyel a `points` értékét is, és ezt a főablak státusz sorába írjuk ki.

> Tipp: A véletlenszám generáláshoz érdemes a `System.Random` objektumot az ablak objektum szintjén példányosítani és nem minden gomb kattintáshoz az erőforrások hatékonyabb használata érdekében.

2.3 Időzítés

Egészítsük ki a programot azzal, hogy számolja és megjeleníti az eltelt időt, a program bezárásakor pedig kiírja, hogy átlagosan hányszor sikerült lenyomni a gombot másodpercenként.



Szükségünk lesz egy `DateTime startTime` privát adattagra, a játékkezdés időbélyegének tárolásához. Használni fogunk egy időzítőt (`System.Windows.Forms.Timer timer`), hogy az elindítása után (`Start()`) megadott időközönként kiváltsa a `Tick` eseményét, amelyhez hozzáösszekötjük a státuszsor frissítését.

A státuszsor tartalmának frissítését szervezzük egy külön eseménykezelő eljárásba és megjelenítjük az eltelt időt is. Két `DateTime` típusú érték különbsége egy `TimeSpan` típusú értéket eredményez, amelynek a `TotalSeconds` tulajdonsága adja meg az eltelt másodpercek számát:

```
private void UpdateStatusBar(object sender, EventArgs e)
{
    double elapsedSeconds = (DateTime.Now - startTime).TotalSeconds;
    statusLabel.Text = $"Points: {points} | Elapsed time: {elapsedSeconds:F0} sec";
}
```

> Az eltelt idő kiírásában az `F0` formázási kifejezéssel 0 tizedesjeggyel írjuk ki az eltelt másodpercek számát.

Ezt az `UpdateStatusBar` eseménykezelőt hívjuk meg a `ButtonClicked` eljárásban, továbbá a `Form1` konstruktorában iratkoztassuk fel a `timer` objektum `Tick` eseményére. Az időzítő `Interval` tulajdonságát állítsuk 1000 ezredmásodpercre.

Az időzítőt az első gombnyomásra indítsuk el, a pontszám növelése helyett:

```
private void ButtonClicked(object sender, EventArgs e)
{
    // ... gomb új helyre pozícionálása ...

    if (!timer.Enabled)
    {
        startTime = DateTime.Now;
        timer.Start();
    }
    else
    {
        ++points;
```

```
    }
    UpdateStatusBar(sender, e);
}
```

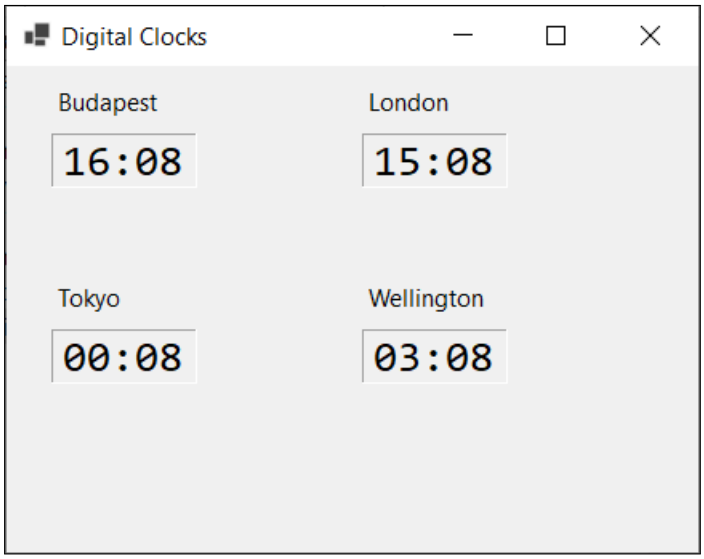
A `Form1` ablak `FormClosing` eseményére is feliratkozunk, amelyet a program bezárása vált majd ki. A programbezárás eseménykezelőjének törzsében létrehozunk egy felugró üzenet-ablakot (`MessageBox`), ami az eddigi sikeres kattintások számát (`points`) elosztja az eltelt másodpercek számával. A statisztika kiírásában két tizedesjegyet engedünk csak meg. A felugró ablakot csak akkor jelenítsük meg, ha már elkezdődött a játék (elindult az időzítő) és az ablak bezárását a felhasználó kezdeményezte (`e.CloseReason`). Utóbbival elkerülhető, hogy a felugró ablak blokkolja az alkalmazás bezárását például az operációs rendszer leállítása esetén.

```
private void GameClosing(object sender, FormClosingEventArgs e)
{
    if (e.CloseReason == CloseReason.UserClosing && timer.Enabled)
    {
        double elapsedSeconds = (DateTime.Now - startTime).TotalSeconds;
        double pushPerSeconds = points / elapsedSeconds;
        MessageBox.Show($"Pushes per seconds: {pushPerSeconds:F2}", "Results",
            MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
}
```

3 Digitális órák különböző időzónákkal ^{OP}

Készítsünk egy olyan alkalmazást, amelyben több - különböző időzónákba eső - város pontos idejét lehet majd egyszerre megjeleníteni. A város nevét és pontos idejét egy saját, egyedi vezérlővel (*user control*) fogjuk megvalósítani.

Készítsünk egy új *Windows Forms App* projektet, amelyhez adjunk egy egyedi vezérlőt a *Solution Explorerben* a projekt nevére jobb egérgombbal kattintva, majd az **Add -> User Control** opciót választva. A vezérlő neve legyen `Clock`. A dialógus ablakot bezárva a Visual Studio el fogja készíteni az osztályt a `UserControl` ősosztályból származtatva. A Visual Studio az ablakokhoz hasonlóan az egyedi vezérlőkhöz is támogatja a grafikus tervezést: a generált kód a `Clock.Designer.cs` fájlba fog kerülni, míg a saját pl. eseménykezelő implementációnkat a `Clock.cs` fájlban adhatjuk meg.



3.1 Egyedi vezérlő elkészítése

A `Clock` vezérlő grafikus felületére adjunk hozzá két szöveges címkét (`Label`) a város és az idő megjelenítésére (`cityLabel`, `timeLabel`). Az időt megjelenítő címke betűtípusát (`Font` tulajdonság) állítsuk

Consolas-ra, a betűméretét növeljük meg és adjunk keretet a vezérlőnek (`BorderStyle` tulajdonság), hogy egy digitális órához hasonlatosabb megjelenést kapjon. Az `AutoSize` tulajdonságot állítsuk hamisra, hogy a vezérlőt ne a tartalma alapján méretezze automatikusan a keretrendszer.

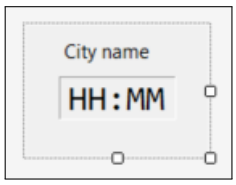


Figure 3: Egyedi vezérlő tervezése

A `Clock`˙vezérlőbe vegyünk fel az időzóna okozta eltérést tartalmazó mezőt (`timeZone`), valamint két tulajdonságot (`City` és `TimeZone`), amivel a város neve és az időeltolódás lekérdezhető / beállítható.

```
private int timeZone;

public string City
{
    get => cityLabel.Text;
    set => cityLabel.Text = value;
}

public int TimeZone
{
    get { return timeZone; }
    set
    {
        timeZone = value;
        RefreshTime(this, EventArgs.Empty);
    }
}
```

A `RefreshTime` metódus a kijelzett idő frissítésért felel és az időt *óra:másodperc* formátumban jeleníti meg, másodpercenként villogó elválasztóval:

```
private void RefreshTime(object sender, EventArgs e)
{
    DateTime time = DateTime.Now;
    timeLabel.Text = time
        .AddHours(TimeZone)
        .ToString(time.Second % 2 == 0 ? "HH:mm" : "HH mm");
}
```

Végezetül adjunk a `Clock` vezérlőhöz egy időzítőt is (`timer`), és az osztály konstruktorában konfiguráljuk úgy, hogy másodpercenként meghívja a `RefreshTime` eseménykezelőt.

3.2 Főablak elkészítése

A `Form1` felületéhez adjunk egy `FlowLayoutPanel` elrendező vezérlőt. Ez a vezérlő a tartalmazott vezérlőket horizontálisan vagy vertikálisan egymás után helyezi el (ha szükséges új sort vagy oszlopot kezdve), így nem szükséges minden tartalmazott vezérlőt külön pozícionálunk. A panel `Dock` tulajdonságát állítsuk `Fill`-re, így kitölti az őt tartalmazó vezérlő (most az ablak) területét.

Az elkészített `Clock` egyedi vezérlő a Windows Forms Designerben is használható, és a *Toolboxból* a megszokott *drag-and-drop* módszerrel a főablak felületére húzható. Adjunk 4 példányt belőle a főablakhoz, a `FlowLayoutPanel`-be.

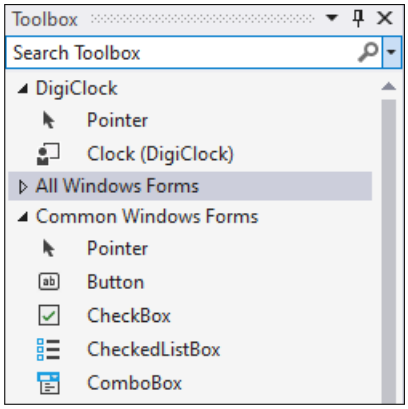


Figure 4: Egyedi vezérlő a *Toolboxban*

> Amennyiben a Toolboxban nem jelenne meg a saját `Clock` vezérlőnk, akkor fordítsuk le a projektet. Ha így sem lenne elérhető, akkor a *Tools* > *Options* > *Windows Forms Designer* > *General* menüpont alatt a *Toolbox* > *AutoToolboxPopulate* beállítás értékét állítsuk igazra.

Valamely `Clock` vezérlő példányt a tervezőben kiválasztva, a *Properties* panelen a `City` és `TimeZone` tulajdonságok értéke beállítható, mivel publikus elemei az osztálynak. Sőt, ha szeretnénk, azt is megmondhatjuk a Visual Studionak attibútumok segítségével, hogy melyik kategórába és milyen magyarázó leírással jelenítse meg őket.

```
[Category("Appearance")]
[Description("The hour offset of the timezone.")]
public int TimeZone { /* ... */ }
```

> Az egyedi vezérlők publikus tulajdonságai elrejthetőek a *Properties* panelről, ha a `[Browsable(false)]` attribútummal annotáljuk.

A 4 `Clock` példánynak adjuk meg 4 főváros nevét és az időzóna okozta eltérést. Például London órája hozzánk képest eggyel hátrébb (-1), Tokióé nyolccal (8), Wellingtoné tizeneggyel (11) előrébb jár.